Reinforcement learning

Consider a dynamic system (not static) and its evolution over time.
We have the concept of states: a state is used to represent the evolution over time of the dynamic system. We also have other elements: observations, state transition model (function that describes how the system evolve over time), noise (used to gather everything that is not explicitly modeled).

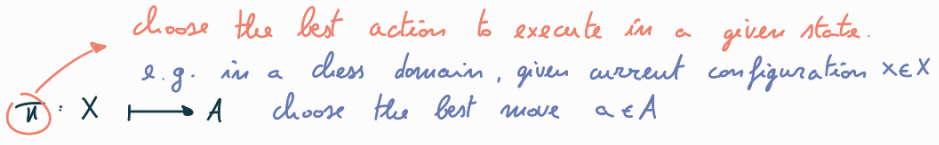
Reasoning: given the model (f, h) and the current state x_k, predict the future (x_{k+T}, z_{k+T})
assuming we have all the necessary information about the dynamic system;
if we know the transition function that tells us what the next state will be given the current one and an action to make, then we don't need to actually make the action. Very often this is not the case.

We cannot predict exactly the future, but we can learn experience by actually executing actions in the environment and try to reconstruct the model.

Learning: given past experience $(z_{0:k})$, determine the model (f, h).

Very important is the concept of state: a state can be seen as a snapshot of the dynamic system.
The state can be either observable or not.

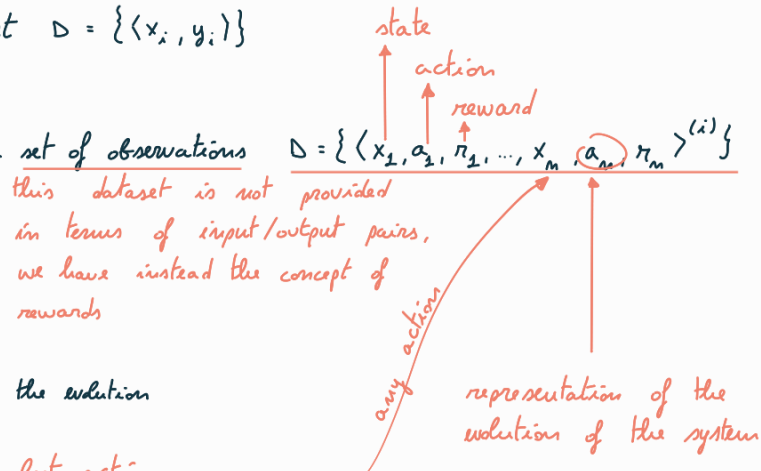
When the state is fully observable, the decision making problem for an agent is to decide which action must be executed in a given state.
The agent has to compute the function



When the model of the system is not known, the agent has to learn the function π.
How to learn the function?

- **Supervised learning**
  learning a function $f : X \to Y$ given a dataset $D = \{(x_i, y_i)\}$

- **Reinforcement learning**
  learning a behavior function $\pi : X \to A$ given a set of observations $D = \{(x_1, a_1, r_1, \dots, x_m, a_m, r_m)^{(i)}\}$



A reward is something given to the agent during the evolution

An action in supervised learning needs to be the best action
An action in reinforcement learning does not need to be the best action
a_m is not the best action we can do in x_m, but is instead the action we actually did in x_m.

Representation of a dynamic system

- X : set of states
  - explicit discrete and finite representation $X = \{x_1, x_2, \dots, x_n\}$
  - continuous representation $x = F(\dots)$ given by a state function
  - probabilistic representation $P(x)$ given by a probabilistic state function
- A : set of actions
  - explicit discrete and finite representation $A = \{a_1, a_2, \dots, a_m\}$
  - continuous representation $A = U(\dots)$ given by a control function
- δ : transition function
  - deterministic
  - non-deterministic
  - probabilistic

  → when an agent executes an action, there will be a probability distribution on what the outcome will be (successor state)
- Z : set of observations
  - explicit discrete and finite representation $Z = \{z_1, \dots, z_k\}$
  - continuous representation $Z = Z(\dots)$ given by an observation function

  → only used in Hidden Markov Models ; here the state is fully observable

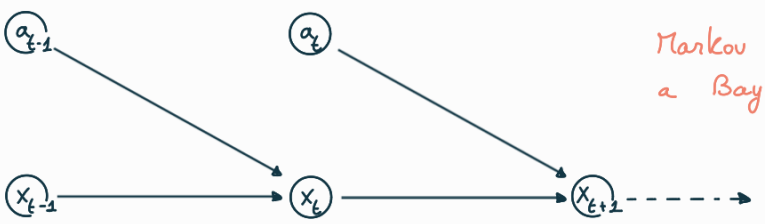
Markov Property

- Once the current state is known, the evolution of the dynamic system does not depend on the history of states, actions and observations ;
- The current state contains all the information needed to predict the future ;
- Future states are conditionally independent of past states and past observations given the current state ;
- The knowledge about the current state makes past, present and future observations statistically independent

A Markov Process is a process that has the Markov property.

if we know the state at time t, the future $(t+1, \dots)$ will not depend on the past

$$P(\text{future} \mid \text{current}, \text{past}) = P(\text{future} \mid \text{current})$$



Markov Process represented by a Bayesian Network

x_{t+1} does not depend on x_{t-1} if we know x_t

Markov Decision Process (MDP)

MDP will model our dynamic system

$$MDP = \langle X, A, \delta, r \rangle$$

In a *deterministic* case:
- X is a finite set of states
- A is a finite set of actions
- $\delta : X \times A \longrightarrow X$ is a transition function
- $r : X \times A \longrightarrow \mathbb{R}$ is a reward function

deterministic transition: any time we execute an action from a state we get the same result

In the *non-deterministic* case:
- X is a finite set of states
- A is a finite set of actions
- $\delta : X \times A \longrightarrow 2^X$ → we get a set of possible states
- $r : X \times A \times X \longrightarrow \mathbb{R}$

different rewards: the reward also depends on the next state

In the *stochastic* case we have a probability distribution associated to the outcome of an action executed from a state

$$\delta(x_t, a_t) \begin{cases} 0.67\% & x_{t+1} = x' \\ 0.23\% & x_{t+1} = x'' \\ 0.05\% & x_{t+1} = x''' \\ 0.05\% & x_{t+1} = x'''' \end{cases}$$

$$\delta = P(x_{t+1} \mid x_t, a_t)$$

Solving an MDP

Solving an MDP means finding a policy (function from states to actions)

$$\pi : X \longrightarrow A$$

We want the *optimal policy*
For each state $x \in X$, $\pi(x) \in A$ is the optimal action to be executed in such state
Optimality criterion = maximize the ——— reward.

A reward says how good the behaviour is for the agent.

- maximizing the *cumulative* reward gives the same importance to short term and long term rewards;
- maximizing the *cumulative discounted* reward is better; the reward collected in the future will be discounted with a discount factor $\gamma \in [0,1]$; this means that the same reward is more valuable now than in the future

value of the policy starting from state x_1: $V^{\pi}(x_1) = E[\, r_1 + \gamma r_2 + \gamma^2 r_3 + \dots \,]$ — Expected value for non-deterministic MDPs

This enforces the agent to gain rewards as soon as possible

$V^{\pi_1}(x_1) > V^{\pi_2}(x_2)$ means that policy π_1 is better than π_2 since it has a higher cumulative reward. The optimal policy is the best one, the one with higher cumulative discounted value.

Optimal policy: $\pi^* = \underset{\pi}{\text{argmax}}\ V^{\pi}(x), \forall x \in X$

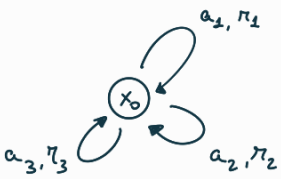
There may exist several optimal policies.

V^* is the value of the optimal policy.

One state MDP

A one-state MDP only has one state : x_0

- x_0
- A is a finite set of actions
- $\delta(x_0, a_i) = x_0 \quad \forall a_i \in A$ transition function
- $r(x_0, a_i, x_0) = r(a_i)$ reward function



Optimal policy : $\pi^*(x_0) = a_i$

This is also called the bandit problem.
This problem is interesting since its solution is a skeleton for most RL problems :

① Initialize a data structure Θ
$\Theta_{(0)}[i] \leftarrow 0$ and $c[i] \leftarrow 0$, $i = 1 \dots |A|$

$\Theta_{(0)}[i]$: i = action index, (0) = time index

② for each time $t = 1, \dots, T$ (until termination condition)

① choose an index $\hat{\imath}$ for action $a_{(t)} = a_{\hat{\imath}} \in A$
② execute action $a_{(t)}$ and collect reward $r_{(t)}$
③ increment $c[\hat{\imath}]$
④ update $\Theta_{(t)}[\hat{\imath}] \leftarrow \frac{1}{c[\hat{\imath}]} \left(r_{(t)} + (c[\hat{\imath}] - 1)\, \Theta_{(t-1)}[\hat{\imath}] \right)$

③ optimal policy : $\pi^*(x_0) = a_{\hat{\imath}}$, with $\hat{\imath} = \text{argmax}_{i=1 \dots |A|} \; \Theta_{(T)}[i]$

How to find the optimal policy ?

- Value iteration (estimate the value function and then compute π)
- Policy iteration (estimate directly π)

Learning through value iteration

The agent could learn the value function $V^{\pi^*}(x)$
From V^{π^*} it can determine the optimal policy:

the optimal policy is the policy that maximizes this quantity

$$\pi^*(x) = \underset{a \in A}{\operatorname{argmax}} \underbrace{[r(x,a) + \gamma V^{\pi^*}(\delta(x,a))]}_{Q(x,a)}$$

However this policy cannot be computed this way because δ and r are not known.
Note that:

$$V^*(x) = \max_{a \in A} \{ r(x,a) + \gamma V^*(\delta(x,a)) \} = \max_{a \in A} Q(x,a)$$

thus we can rewrite

transition function

$$Q(x,a) = r(x,a) + \gamma V^*(\delta(x,a))$$

as:

$$Q(x,a) = r(x,a) + \gamma \max_{a'} (Q(\delta(x,a), a'))$$

Instead of using the real Q function, we use an estimate $\hat{Q}$ (approximation of $Q(x,a)$ at time t)

$$\hat{Q}(x,a) \longleftarrow \bar{r} + \gamma \max_{a'} \hat{Q}(x',a')$$

$\bar{r}$: immediate reward we get when we execute a from x

$\max_{a'} \hat{Q}(x',a')$: estimate

$\hat{Q}$ underestimates Q!

$$0 \leq \hat{Q}_{(t)}(x,a) \leq \hat{Q}_{(t+1)}(x,a) \leq Q(x,a)$$

We have convergence

At the next iteration we will have an estimate $\hat{Q}(x,a)$
We don't know the reward function until we execute the action

Algorithm:

1. for each x, a initialize table entry $\hat{Q}_{(0)}(x,a) \longleftarrow 0$
2. observe current state x
3. for each time $t = 1, \dots, T$ (until termination condition)
   - choose an action a
   - execute the action a
   - observe the new state x'
   - collect immediate reward $\bar{r}$
   - update the table entry for $\hat{Q}(x,a)$ as follows:

$$\hat{Q}_{(t)}(x,a) \longleftarrow \bar{r} + \gamma \max_{a' \in A} \hat{Q}_{(t-1)}(x',a')$$

   - $x \longleftarrow x'$
4. Optimal policy: $\pi^*(x) = \underset{a \in A}{\operatorname{argmax}} \hat{Q}_{(T)}(x,a)$

Note: not using δ and r, but just observing new state x' and immediate reward $\bar{r}$ after the execution of the chosen action

How actions are chosen by the agents ?

- Exploitation : select action a that maximizes $\hat{Q}(x,a)$
- Exploration : select random action a (with low value of $\hat{Q}(x,a)$)

ε-greedy strategy

Given $0 \leq \varepsilon \leq 1$

- select a random action with probability ε
- select the best action with probability $1-\varepsilon$

ε can decrease over time (first exploration, then exploitation)

Soft-max strategy

Actions with higher $\hat{Q}$ values are assigned higher probabilities, but every action is assigned a non-zero probability

↳ in order to have convergence

$$P(a_i|x) = \frac{K^{\hat{Q}(x,a_i)}}{\sum_j K^{\hat{Q}(x,a_j)}}$$

$K > 0$ determines how strongly the selection favors actions with high $\hat{Q}$ values.
K may increase over time (first exploration, then exploitation).